Efficient Algorithms for Approximate Triangle Counting

Mostafa Haghir Chehreghani

Department of Computer Science,
Katholieke Universiteit Leuven, Belgium
Mostafa.HaghirChehreghani@cs.kuleuven.be

Abstract. Counting the number of triangles in a graph has many important applications in network analysis. Several frequently computed metrics like the clustering coefficient and the transitivity ratio need to count the number of triangles in the network. Furthermore, triangles are one of the most important graph classes considered in network mining. In this paper, we present a new randomized algorithm for approximate triangle counting. The algorithm can be adopted with different sampling methods and give effective triangle counting methods. In particular, we present two sampling methods, called the *q-optimal sampling* and the *edge sampling*, which respectively give $O(sm)$ and $O(sn)$ time algorithms with nice error bounds (m and n are respectively the number of edges and vertices in the graph and s is the number of samples). Among others, we show, for example, that if an upper bound $\widetilde{\Delta^e}$ is known for the number of triangles incident to every edge, the proposed method provides an $1 \pm \epsilon$ approximation which runs in $O(\frac{\widetilde{\Delta^e} n \log n}{\widehat{\Delta^e} \epsilon^2})$ time, where $\widehat{\Delta^e}$ is the average number of triangles incident to an edge. Finally we show that the algorithm can be adopted with streams. Then it, for example, will perform 2 passes over the data (if the size of the graph is known, otherwise it needs 3 passes) and will use $O(sn)$ space.

Keywords: Graphs, triangles, approximate algorithms, stream data, network analysis, complexity.

1 Introduction

Graphs are fundamental structures for modeling complex relationships between data. Examples include: Internet (where vertices are routers and edges correspond to physical links), World Wide Web (where vertices are web pages and edges correspond to hyperlinks), social networks (where vertices are humans and edges correspond to friendships), traffic data (where vertices are places or cities and edges correspond to roads) and biological networks (where vertices are proteins and edges correspond to protein interactions).

The problem of counting subgraphs of a certain class, is one of the typical problems in graph mining which in recent years has obtained considerable attentions. Triangles are one of the most important basic subgraphs. On the other hand, computation of several network indices and statistics are based on counting the number of triangles, which makes triangle counting an essential problem in network analysis. *Clustering coefficient* of a graph [18] is defined as the normalized sum of the fraction of neighbor pairs of a

vertex of the graph that are connected. *Transitivity coefficient* of a graph [11], is defined as the ratio between three times the number of triangles and the number of length two paths in the graph.

In terms of time complexity, the most efficient triangle counting algorithms are based on matrix multiplication. If A is the adjacency matrix of a graph G, the number of triangles in G is equal to

$$\frac{1}{6}\mathbf{Tr}(A^3) \tag{1}$$

where $\mathbf{Tr}(\cdot)$ denotes the *trace* of a matrix defined as the sum of the elements on the main diagonal of the matrix.

Time complexity of the most efficient known algorithm for matrix multiplication is $O(n^{2.3727})$ [6], [19], where n is the number of vertices of the graph (the number of rows/columns of A). The exponent of n, denoted by ω, is called *matrix multiplication exponent*. Alon et al. give in [1] a more efficient triangle counting algorithm for sparse graphs. Time complexity of their method is $O(m^{\frac{2\omega}{\omega+1}})$, where m is the number of edges of the graph. For $\omega = 2.3727$, this time complexity is equal to $O(m^{1.41})$.

However, exact triangle counting methods may be inefficient when the size of the graph is large. In these cases, an approximate algorithm is preferred in the cost of losing the exact number of triangles. In recent years, many algorithms have been proposed for approximate triangle counting. A widely used technique is the *sparsification technique* [17], [16], [14] and [13]. In this technique, the graph is converted into a sparse graph and the number of triangles in the sparsified graph is counted. Then, the result is scaled to the original graph. Some other methods are based on approximate computation of algebraic properties of the graph like *eigenvalues* of the adjacency matrix [15]. However, well-known methods for computing (or approximating) eigenvalues are heavily based on matrix multiplication and it is known that worst case time complexity of computing eigenvalues is the same as matrix multiplication. On the hand, such approximate triangle counting algorithms look at the algebraic methods (like approximate matrix multiplication and low rank matrix approximation) as a black-box. However, samplings in the algebraic methods are done in a way to minimize the element-wise error or the frobenius norm of the error matrix and therefore, the error of triangle counting is not minimized.

In this paper, we propose a new randomized algorithm for approximate triangle counting. Our method is a variation of several approximate matrix multiplication algorithms [8], [10] and [9]. However, it does not generate any product matrix and several algorithmic aspects are different. Furthermore, the sampling methods which are crucial elements of the algorithm, are also different. The proposed algorithm can be seen as a general framework to which different sampling methods can be applied. Every sampling method gives a new triangle counting algorithm with its own error bounds and time complexity. In particular, we present two sampling methods, called the *q-optimal sampling* and the *edge sampling*, which respectively give $O(sm)$ and $O(sn)$ time algorithms with nice error bounds (m and n are respectively the number of edges and vertices in the graph and s is the number of samples). Among others, we show, for example, that if an upper bound $\widetilde{\Delta^e}$ is known for the number of triangles incident to every edge, the proposed method provides an $1 \pm \epsilon$ approximation which runs in $O(\frac{\widetilde{\Delta^e} n \log n}{\widetilde{\Delta^e} \epsilon^2})$

time, where $\widehat{\Delta^e}$ is the average number of triangles incident to an edge. As we will discuss, some existing algorithms can be seen as adoptations of the proposed algorithm with specific sampling methods. We finally show that the algorithm can be extended to streams. Then it, for example, will perform 2 passes over the data (if the size of the graph is known, otherwise it needs 3 passes) and will use $O(sn)$ memory cells.

The rest of this paper is organized as follows. In Section 2, we present a new randomized algorithm for approximate triangle counting. In Section 3, different sampling methods are introduced. In Section 4 we extend the algorithm for counting triangles in streams. An overview of related work is provided in Section 5. Finally, the paper is concluded in Section 6.

Throughout the paper, G refers to a simple (i.e. loop-free and without multiple edges) and undirected graph. A refers to the adjacency matrix of G. Therefore, A is a square matrix consisting of 0s and 1s. A_{ij} denotes the element in the i-th row and the j-th column of A. n and m denote the number of vertices of G (the number of rows and the number of columns of A) and the number of edges of G, respectively. Δ denotes the number of triangles in G. In this paper, we use an index i for referring to a row (column) in the adjacency matrix as well as for referring to the vertex of the graph corresponds to the row (column) i.

2 The approximate triangle counting algorithm

In this section, we present a randomized algorithm for approximate triangle counting. Suppose A is an $n \times n$ matrix which is the adjacency matrix of a graph G. As Equation 1 shows, in order to count triangles in G, we can compute the trace of A^3. Algorithm 1 shows the high level pseudo code of an approximate triangle counting algorithm, based on randomized calculation of the trace of A^3.

In every iteration t of the loop in Lines 3-10 of Algorithm 1, first the following probabilities are computed:

$$p_1, p_2, \ldots, p_n \geq 0 \text{ such that } \sum_{i=1}^{n} p_i = 1$$

Then, an $i \in \{1, \ldots, n\}$ is selected with probability p_i, and the following probabilities are computed:

$$q_{1|i}, \ldots, q_{n|i} \geq 0 \text{ such that } \sum_{j=1}^{n} q_{j|i} = 1$$

Finally, an $j \in \{1, \ldots, n\}$ is selected with probability $q_{j|i}$ and the number of triangles is estimated by

$$\beta_t = \frac{\sum_{d=1}^{n} A_{di} A_{ij} A_{jd}}{6 p_i q_{j|i}} \tag{2}$$

The final estimation, β, is the average of the estimations of different trials.

In the rest of this section, we study some important properties of Algorithm 1.

Algorithm 1 High level pseudo code of the approximate triangle counting algorithm.

TRIANGLECOUNTER

Require: A graph G.
Ensure: The approximate number of triangles in G.
1: {Let A be the $n \times n$ adjacency matrix of G}
2: $\beta \leftarrow 0$
3: **for** $t = 1$ **to** s **do**
4: Compute $p_1, \ldots, p_n$
5: Select $i \in \{1, \ldots, n\}$ with the probability p_i
6: Compute $q_{1|i}, \ldots, q_{n|i}$
7: Select $j \in \{1, \ldots, n\}$ with the probability $q_{j|i}$
8: $\beta_t \leftarrow \frac{\sum_{d=1}^{n} A_{di} A_{ij} A_{jd}}{6 p_i q_{j|i}}$
9: $\beta \leftarrow \beta + \beta_t$
10: **end for**
11: **return** $\frac{\beta}{s}$

Lemma 1. *In Algorithm 1, for every $t \in \{1, \ldots, s\}$ we have:*

$$\mathbf{E}(\beta_t) = \mathbf{E}(\beta) = \frac{\mathbf{Tr}(A^3)}{6} \tag{3}$$

Proof. For every $t \in \{1, \ldots, s\}$, we have:

$$\begin{aligned}
\mathbf{E}(\beta_t) &= \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\frac{\sum_{d=1}^{n} A_{di} A_{ij} A_{jd}}{6 p_i q_{j|i}} p_i p_{j|i} \right) \\
&= \frac{1}{6} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{d=1}^{n} A_{di} A_{ij} A_{jd} \\
&= \frac{1}{6} \mathbf{Tr}(A^3)
\end{aligned}$$

On the other hand, β_t's are independent random variables and β is the sum of s independent random variables $\beta_1 \ldots \beta_s$ divided by s. Therefore

$$\mathbf{E}(\beta) = \frac{\mathbf{E}(\sum_{t=1}^{s} \beta_t)}{s} = \frac{s \mathbf{E}(\beta_t)}{s} = \mathbf{E}(\beta_t) = \frac{\mathbf{Tr}(A^3)}{6}$$

■

Lemma 2. *In Algorithm 1, variance of every random variable β_t is*

$$\mathbf{Var}(\beta_t) = \frac{1}{36} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\left(\sum_{d=1}^{n} A_{di} A_{ij} A_{jd} \right)^2}{p_i q_{j|i}} - \left(\mathbf{Tr}(A^3) \right)^2 \right) \tag{4}$$

Proof. We have:

$$\mathbf{Var}(\beta_t) = \mathbf{E}(\beta_t^2) - \left(\mathbf{E}(\beta_t) \right)^2$$

Then

$$\mathbf{E}(\beta_t^2) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{\sum_{d=1}^{n} A_{di}A_{ij}A_{jd}}{6p_iq_{j|i}}\right)^2 p_ip_{j|i} = \frac{1}{36}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\left(\sum_{d=1}^{n} A_{di}A_{ij}A_{jd}\right)^2}{p_iq_{j|i}} \tag{5}$$

and

$$\left(\mathbf{E}(\beta_t)\right)^2 = \frac{1}{36}\left(\mathbf{Tr}(A^3)\right)^2 \tag{6}$$

Therefore

$$\mathbf{Var}(\beta_t) = \frac{1}{36}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\left(\sum_{d=1}^{n} A_{di}A_{ij}A_{jd}\right)^2}{p_iq_{j|i}} - \left(\mathbf{Tr}(A^3)\right)^2\right) \tag{7}$$

■

Since β is the average of s independent copies of β_t, then

$$\mathbf{Var}(\beta) = \frac{\mathbf{Var}(\beta_t)}{s} = \frac{1}{36s}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\left(\sum_{d=1}^{n} A_{di}A_{ij}A_{jd}\right)^2}{p_iq_{j|i}} - \left(\mathbf{Tr}(A^3)\right)^2\right) \tag{8}$$

For a vertex i of the graph, *local triangles* of i are triangles which are incident to i. The number of local triangles of i, denoted by Δ_i, equals to

$$\Delta_i = \frac{1}{2}\sum_{j=1}^{n}\sum_{d=1}^{n} A_{di}A_{ij}A_{jd}$$

Let $\{i,j\}$ be an edge of the graph. *Local triangles* of $\{i,j\}$ are triangles for which $\{i,j\}$ is an edge. The number of local triangles of $\{i,j\}$, denoted by $\Delta_{\{i,j\}}$, is equal to

$$\Delta_{\{i,j\}} = \sum_{d=1}^{n} A_{di}A_{ij}A_{jd}$$

If i is not connected to j, the number of local triangles of $\{i,j\}$ is equal to 0. The following holds between Δ_i and $\Delta_{\{i,j\}}$: $\Delta_i = \frac{1}{2}\sum_{j=1}^{n}\Delta_{\{i,j\}}$. Let Δ refer to the number of triangles in the graph. We have: $\Delta = \frac{1}{3}\sum_{i=1}^{n}\Delta_i$.

Using the notion of local triangles of edges, Equation 8 can be re-written as:

$$\mathbf{Var}(\beta) = \frac{1}{36s}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\Delta_{\{i,j\}}^2}{p_iq_{j|i}} - \frac{\Delta^2}{s} \tag{9}$$

Lemma 3. *If in Algorithm 1 probabilities $p_1, p_2, \ldots, p_n$ and $q_{1|i}, q_{2|i}, \ldots, q_{n|i}$ are accessible in constant time, its time complexity will be $O(sn)$.*

Proof. The loop in Lines 3-10 is performed for s times. Inside the loop, the most time consuming step is Line 8 which takes $O(n)$ time. Therefore, time complexity of the algorithm is $O(sn)$. ■

3 Sampling methods

In this section, we present a number of sampling methods. First in Section 3.1 the optimal sampling is investigated. Then, since the optimal sampling might be computationally expensive, other near-optimal samplings are introduced.

3.1 Optimal sampling

Lemma 4 introduces the probabilities and error bound of the optimal sampling.

Lemma 4. *If for $1 \leq i \leq n$, p_i's are equal to*

$$p_i = \frac{\Delta_i}{3\Delta} \tag{10}$$

and then after selecting i, for $1 \leq j \leq n$, $q_{j|i}$'s are

$$q_{j|i} = \frac{\Delta_{\{i,j\}}}{2\Delta_i} \tag{11}$$

the variance of β presented in Equation 8 is minimized. The minimized variance is 0.

Since the error bound of the optimal sampling is 0, it gives an exact triangle counting algorithm. On the other hand, time complexity of computation of p_i's in Equation 10 is the same as time complexity of exact triangle counting. Therefore, using Algorithm 1 with the optimal sampling is the same (in both accuracy and complexity) as using an exact triangle counting algorithm.

3.2 q-optimal sampling

In the q-optimal sampling, every vertex i, $1 \leq i \leq n$, is selected by some strategy (which can be performed in $O(1)$ time). For example, they are selected uniformly at random (therefore $p_i = \frac{1}{n}$), or they are selected proportional to their degrees (therefore, $p_i = \frac{\mathbf{deg}(i)}{2m}$, where $\mathbf{deg}(i)$ refers to the degree of i). Then, every vertex j is selected in a way to minimize $\mathbf{Var}(\beta)$.

Lemma 5. *In the q-optimal sampling, after choosing a vertex i, if every vertex j is selected with probability*

$$q_{j|i} = \frac{\Delta_{\{i,j\}}}{2\Delta_i} \tag{12}$$

the variance of β is minimized.

Proof. In order to minimize $\mathbf{Var}(\beta)$, we need to minimize $\sum_{j=1}^{n} \frac{{\Delta_{\{i,j\}}}^2}{q_{j|i}}$, because other parts of $\mathbf{Var}(\beta)$ are independent of j. We define

$$f(q_{1|i}, q_{2|i}, \ldots, q_{n|i}) = \sum_{j=1}^{n} \frac{{\Delta_{\{i,j\}}}^2}{q_{j|i}}$$

and substitute $q_{n|i}$ by $1-\sum_{j'=1}^{n-1} q_{j'|n}$ and form equations $\frac{\partial f}{\partial q_{j|i}} = 0$, for $1 \leq j \leq n-1$.

We get

$$\frac{{\Delta_{\{i,j\}}}^2}{{q_{j|i}}^2} = \frac{{\Delta_{\{i,n\}}}^2}{\left(1-\sum_{j'=1}^{n-1} q_{j|i}\right)^2}$$

$$\Rightarrow q_{j|i} = \frac{1-\sum_{j'=1}^{n-1} q_{j|i}}{\Delta_{\{i,n\}}} \Delta_{\{i,j\}} \tag{13}$$

Summing $q_{j|i}$'s, for $1 \leq j \leq n-1$, and doing simplifications, we get

$$\sum_{j=1}^{n-1} q_{j|i} = \frac{\sum_{j'=1}^{n-1} \Delta_{\{i,j\}}}{\sum_{j'=1}^{n} \Delta_{\{i,j\}}}$$

Putting the value of $\sum_{j=1}^{n-1} q_{j|i}$ into Equation 13 and doing simplifications, we get the value of $q_{j|i}$ for the q-optimal sampling:

$$q_{j|i} = \frac{\Delta_{\{i,j\}}}{\sum_{j'=1}^{n} \Delta_{\{i,j'\}}} = \frac{\Delta_{\{i,j\}}}{2\Delta_i}$$

■

If vertices i are selected proportional to their degrees, the variance of β in the q-optimal sampling will be

$$\mathbf{Var}(\beta) = \frac{2m}{9s} \sum_{i=1}^{n} \frac{{\Delta_i}^2}{\mathbf{deg}(i)} - \frac{\Delta^2}{s} \tag{14}$$

and if they are selected selected uniformly at random, $\mathbf{Var}(\beta)$ will be

$$\mathbf{Var}(\beta) = \frac{n}{9s} \sum_{i=1}^{n} {\Delta_i}^2 - \frac{\Delta^2}{s} \tag{15}$$

The motivation for selecting vertices i proportional to their degrees is that, as studied in [15], in most of real-world networks, vertices of higher degrees have higher number of local triangles. Then, for every two vertices i and i', if it holds that $deg(i) \geq deg(i')$ implies $\Delta_i \geq \Delta_{i'}$, it can be shown that selecting vertices i proportional to their degrees gives a better sampling than choosing them uniformly at random.

If the q-optimal sampling is used, in every iteration of the loop in Lines 3-10 of Algorithm 1, probabilities p_i and $q_{j|i}$ can be computed in $O(m)$ time. On the other hand, it takes $O(n)$ time to compute β_t. Therefore, time complexity of Algorithm 1 with the q-optimal sampling will be $O(sm)$.

The q-optimal sampling can provide efficient $1 \pm \epsilon$ approximations, specifically if some information on local triangles of *vertices* is available. For example, consider the version of the q-optimal sampling where vertices i are selected uniformly at random

($p_i = \frac{1}{n}$). Suppose that there exists an already known value $\widetilde{\Delta^v}$ such that for every vertex i, $\Delta_i \leq \widetilde{\Delta^v}$. Then, in every iteration of the loop in Lines 3-10 of Algorithm 1, a random variable X_t can be defined as $X_t = \frac{\beta_t}{n\widetilde{\Delta^v}} = \frac{\Delta_{i(t)}}{3\widetilde{\Delta^v}}$, where $\Delta_{i(t)}$ is the number of local triangles of the vertex selected in iteration t. We have: $\mathbf{E}(X_t) = \frac{\Delta}{n\widetilde{\Delta^v}} = \frac{\widehat{\Delta^v}}{\widetilde{\Delta^v}}$, where $\widehat{\Delta^v}$ is the average number of local triangles of vertices. By Chernoff bound we obtain:

$$\mathbf{Pr}\left[\frac{1}{s}\sum_{t=1}^{s} X_t - \frac{\widehat{\Delta^v}}{\widetilde{\Delta^v}} > \epsilon\frac{\widehat{\Delta^v}}{\widetilde{\Delta^v}}\right] \leq \exp\left(-\frac{\epsilon^2 s\widehat{\Delta^v}}{2\widetilde{\Delta^v}}\right) \tag{16}$$

If $s = \Omega(\frac{\widetilde{\Delta^v}\log n}{\widehat{\Delta^v}\epsilon^2})$, then $\frac{n\widetilde{\Delta^v}}{s}\sum_{t=1}^{s} X_t$ approximates Δ within a factor of ϵ with probability at least $1 - n^{-c}$ for any constant c. This gives an $O(\frac{\widetilde{\Delta^v} m\log n}{\widehat{\Delta^v}\epsilon^2})$ time algorithm which approximates Δ within a factor of ϵ. Specifically, if $\widetilde{\Delta^v}$ is greater than $\widehat{\Delta^v}$ only by a factor of a constant, time complexity of the algorithm will be $O(\frac{m\log n}{\epsilon^2})$.

3.3 Edge sampling

In the edge sampling, first a vertex i is selected by some strategy (which can be done in $O(1)$ time). Then, a neighbor j of i is selected by some (probably different) strategy which also can be done in $O(1)$ time. Since in this sampling computation of probabilities is done in $O(1)$ time, time complexity of the algorithm is $O(sn)$.

For example, i can be selected uniformly at random (therefore, $p_i = \frac{1}{n}$), then, for every j, if j is a neighbor of i, $q_{j|i}$ is equal to $\frac{1}{\mathbf{deg}(i)}$; otherwise it is 0. This case is similar to the methods which uniformly sample an edge and count the number of triangles incident to it and scale the result. The first algorithm proposed in [12] and partially the algorithm of [5] are examples of such methods. In this case, variance of β will be

$$\mathbf{Var}(\beta) = \frac{n}{36s}\sum_{i=1}^{n}\left(\mathbf{deg}(i)\sum_{j=1}^{n}{\Delta_{\{i,j\}}}^2\right) - \frac{\Delta^2}{s} \tag{17}$$

In the second case of the edge sampling, i is chosen with probability $p_i = \frac{\mathbf{deg}(i)}{2m}$. Then, similar to the first case, for every vertex j, if j is a neighbor of i, $q_{j|i}$ will be $\frac{1}{\mathbf{deg}(i)}$. Otherwise, it will be 0. Variance of β in this case is:

$$\mathbf{Var}(\beta) = \frac{m}{18s}\sum_{i=1}^{n}\sum_{j=1}^{n}{\Delta_{\{i,j\}}}^2 - \frac{\Delta^2}{s} \tag{18}$$

Similar to the q-optimal sampling, the edge sampling can provide efficient $1 \pm \epsilon$ approximations, specifically if some information on local triangles of *edges* is available. For example, consider the second case and suppose that there exists a known value $\widetilde{\Delta^e}$ such that for every edge $\{i,j\}$, $\Delta_{\{i,j\}} \leq \widetilde{\Delta^e}$. Then, in every iteration of the loop in Lines 3-10 of Algorithm 1, a random variable X_t can be defined as $X_t = \frac{\beta_t}{m\widetilde{\Delta^e}} =$

$\frac{\Delta_{\{i,j\}(t)}}{3\widehat{\Delta^e}}$, where $\Delta_{\{i,j\}(t)}$ is the number of local triangles of the edge $\{i,j\}$ selected in iteration t. We have: $\mathbf{E}(X_t) = \frac{\Delta}{m\widehat{\Delta^e}} = \frac{\widehat{\Delta^e}}{\widehat{\Delta^e}}$, where $\widehat{\Delta^e}$ is the average number of local triangles of edges. Similarly, by Chernoff bound we obtain:

$$\mathbf{Pr}\left[\frac{1}{s}\sum_{t=1}^{s} X_t - \frac{\widehat{\Delta^e}}{\widehat{\Delta^e}} > \epsilon\frac{\widehat{\Delta^e}}{\widehat{\Delta^e}}\right] \leq \exp\left(-\frac{\epsilon^2 s\widehat{\Delta^e}}{2\widehat{\Delta^e}}\right) \tag{19}$$

and if $s = \Omega(\frac{\widetilde{\Delta^e}\log n}{\widehat{\Delta^e}\epsilon^2})$, then $\frac{m\widetilde{\Delta^e}}{s}\sum_{t=1}^{s} X_t$ approximates Δ within a factor of ϵ with probability at least $1-n^{-c}$ for any constant c. This gives an $O(\frac{\widetilde{\Delta^e} n\log n}{\widehat{\Delta^e}\epsilon^2})$ time algorithm which approximates Δ within a factor of ϵ. Specifically, if $\widetilde{\Delta^e}$ is greater than $\widehat{\Delta^e}$ only by a factor of a constant, time complexity of the algorithm will be $O(\frac{n\log n}{\epsilon^2})$.

4 Triangle counting in streams

In many applications like World Wide Web and social networks, the dataset is too large to load it into the main memory. A widely used approach to address this problem is the use of the data stream model. A data stream is an ordered sequence in which data arrives one item at a time, and the algorithm has access to limited computation and storage capabilities.

In this section, we extend the algorithm presented in Section 2 to streams. While our focus is the q-optimal sampling where vertices i are selected uniformly at random, the other sampling methods can be extended to streams in a similar way. Due to lack of space, we here omit details. If the number of vertices of the graph, n, is already known, our algorithm will need 2 passes over the data. Otherwise, an extra pass will be needed to find it. First, s integers (vertices) i are selected independently at random with uniform probability $\frac{1}{n}$. Then:

- During the first pass, for every i, the *neighborhood vector* of i, denoted by $\mathcal{I}^i$, is formed. The neighborhood vector of i, shows which vertices of the graph are a neighbor of i and which ones are not. For every vertex j, if $\{i,j\}$ is an edge of the graph, ${\mathcal{I}^i}_j$ is set to 1, otherwise, it is set to 0.
- During the second pass, for every vertex i and for all vertices j, the number of local triangles of the edge between i and j is calculated. To do so, for every i a vector $\mathcal{P}^i$ of size n is used, where ${\mathcal{P}^i}_j$ stores the number of local triangles of $\{i,j\}$. When an edge A_{jd} is visited, if there exists an edge between i and j (i.e. ${\mathcal{I}^i}_j = 1$) and an edge between i and d (i.e. ${\mathcal{I}^i}_d = 1$), a triangle consisting of the vertices i, j and d is found. Therefore, ${\mathcal{P}^i}_j$ and ${\mathcal{P}^i}_d$ and z^i are increased by 1. z^i is used to store the number of local triangles of i.

During these two passes, the information required for q-optimal sampling and approximate triangle counting are gathered. Then, for every i, a $j \in \{1,\ldots,n\}$ is selected with probability $q_{j|i} = \frac{\Delta_{\{i,j\}}}{2\Delta_i} = \frac{\mathcal{P}_j}{2z^i}$ and the approximate number of triangles is calculated.

Every pass needs $O(sn)$ space from the main memory. After twice passing over the stream and calculating $\mathcal{I}^i$s, $\mathcal{P}^i$s and z^is, the approximate number of triangles can be

determined in $O(1)$ time. We can store random variables X_t, if $\widetilde{\Delta^v}$ is known. In this case, we will need $O(\frac{\widetilde{\Delta^v} n \log n}{\widetilde{\Delta^v} \epsilon^2})$ space to provide a $1 \pm \epsilon$ approximation. Therefore, we can present the following theorem:

Theorem 1. *There is a 2-pass algorithm (if n is already known, otherwise it will need 3 passes) to count the number of triangles in a stream of edges which needs $O(sn)$ memory space and constant update time and its error guarantee obeys Equation 15. Specifically, with space usage $O(\frac{\widetilde{\Delta^v} n \log n}{\widetilde{\Delta^v} \epsilon^2})$, it gives a $1 \pm \epsilon$ approximation.*

Similar results can be presented for other samplings.

5 Related work

Buriol et al. [5] presented one of the first approximate triangle counting algorithms. In their method, an edge and a vertex are selected by random and it is checked whether they form a triangle or not. Then, the fraction of tests which form a triangle is scaled and returned as an estimation of the number of triangles. In this method, if

$$s = \mathbf{log}(\frac{1}{\delta}) \frac{1}{\epsilon^2} \left(\frac{T_0 + T_1 + T_2 + T_3}{T_3} \right)$$

independent trials are done, where T_i is the number of triples of vertices with i edges, with probability at least $1 - \delta$, the estimated number of triangles is between $(1 - \epsilon)T_3$ and $(1 + \epsilon)T_3$. However, $T_0 + T_1 + T_2$ can be very large compared to T_3.

The other method which is efficient only for triangle-dense graphs is [2]. Their method is based on reducing triangle counting to estimating the zero-th, first and second frequency moments. They showed that there is a streaming algorithm that for any adjacency stream of a graph, computes an $(\epsilon - \delta)$ approximation of the number of triangles using space:

$$O\left(\frac{1}{\epsilon^3} \times \mathbf{log} \frac{1}{\delta} \times \left(\frac{T_1 + T_2 + T_3}{T_3} \right)^3 \times \mathbf{log} n \right)$$

Tsourakakis [15] proposed a triangle counting algorithm based on computing the largest eigenvalues of the adjacency matrix of an undirected graph to approximate both the total as well as the local number of triangles in the graph. Tsourakis's method exploits the following property: the total number of triangles in an undirected graph is $\frac{1}{6} \sum_{j=1}^{n} {\lambda_j}^3$, where λ_j is the j-th eigenvalue of A. Using the Lanczos method [7], the eigenvalues are generated from the biggest one to the smallest one. An approximation is done when the smallest generated eigenvalue contributes very little to the total number of triangles. However, it is known that time complexity of finding eigenvalues is the same as time complexity of matrix multiplication. For approximating the number of local triangles of a vertex i, he exploited the following property: $\Delta_i = \frac{\sum_{j=1}^{n} {\lambda_j}^3 {u_{i,j}}^2}{2}$, where λ_j is the j-th eigenvalue and $u_{i,j}$ is the j-th entry of the i-th eigenvector of A.

In [16], the authors proposed the DOULION algorithm for approximate triangle counting. In this method, every edge e of the graph is removed with a sparsification

probability $1-p$. If it survives, the weight $\frac{1}{p}$ is assigned to it. Then, the number of triangles in the original graph is approximated as the number of triangles in the sparsified graph times $\frac{1}{p^3}$. The following error bound was provided for this method:

$$\mathbf{Var}(\Delta) = \frac{\Delta(p^3 - p^6) + 2k(p^5 - p^6)}{p^6}$$

where k is the number of pairs of triangles which are edge-disjoint.

The randomized algorithm of [14] colors the vertices of the graph with $N = \frac{1}{p}$ colors uniformly at random, counts triangles whose vertices have the same color, and scales that count appropriately. The authors showed that for enough large values of p, their estimation of the number of triangles is concentrated around its expectation. In [13], the authors combined the sparsification techniques and the the idea of vertex partitioning (into *high degrees* and *low degrees*) presented in [1]. They developed an $O(m + \frac{m^{\frac{3}{2}} \log n}{\Delta \epsilon^2})$ time $(1 \pm \epsilon)$-approximation algorithm.

In [3], the authors studied the problem of approximate local triangle counting in large graphs. Their approximation algorithms are based on the idea of min-wise independent permutations [4]. Their algorithms operate in a semi-streaming fashion, using $O(n)$ space in main memory and performing $O(\mathbf{log}n)$ passes over the edges of the graph.

6 Conclusions

In this paper, we proposed a new randomized algorithm for approximate triangle counting. The algorithm provides a general framework which can be adopted with different sampling techniques and give methods with different time complexities and error bounds. For example, it can be adopted with the q-*optimal sampling* and the *edge sampling*, presented in the paper, and give linear time algorithms for approximate triangle counting. We showed that if an upper bound $\widetilde{\Delta^e}$ is known for the number of triangles incident to every edge, the proposed method provides an $1 \pm \epsilon$ approximation which runs in $O(\frac{\widetilde{\Delta^e} n \log n}{\widehat{\Delta^e} \epsilon^2})$ time, where $\widehat{\Delta^e}$ is the average number of triangles incident to an edge. Also, if an upper bound $\widetilde{\Delta^v}$ is known for the number of triangles incident to every vertex, the proposed method provides an $1 \pm \epsilon$ approximation which runs in $O(\frac{\widetilde{\Delta^v} m \log n}{\widehat{\Delta^v} \epsilon^2})$ time, where $\widehat{\Delta^v}$ is the average number of triangles incident to a vertex.

Finally we showed the algorithm can be extended to streams. Then it, for example, will perform 2 passes over the data (if the size of the graph is known, otherwise it needs 3 passes) and will use $O(sn)$ space.

References

1. Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
2. Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632. ACM, 2002.

3. Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(3):1–28, 2010.
4. A. Z. Broder. On the resemblance and containment of documents. In *In Proceedings of the Compression and Complexity of Sequences*, pages 21–29. IEEE, 1998.
5. Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *In Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 253–262. ACM, 2006.
6. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.
7. Jane K. Cullum and Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1: Theory*. Classics in Applied Mathematics 41, SIAM, 2002.
8. Petros Drineas, Alan M. Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay. Clustering in large graphs and matrices. In *SODA*, pages 291–299. ACM, 1999.
9. Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM J. Comput.*, 36(1):132–157, 2006.
10. Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM J. Comput.*, 36(1):158–183, 2006.
11. Frank Harary and Helene J. Kommel. Matrix measures for transitivity and balance. *Journal of Mathematical Sociology*, 6:199–210, 1979.
12. Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *COCOON*, pages 710–716. Springer, 2005.
13. Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.
14. Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Inf. Process. Lett.*, 112(7):277–281, 2012.
15. Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *In Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 608–617. IEEE, 2008.
16. Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *KDD*, pages 837–846, 2009.
17. Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011.
18. Duncan J. Watts and Steven H. Strogatz. Dynamics of small-world networks. *Nature*, 393:440–442, 1998.
19. Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898. ACM, 2012.